Zach North

603885768

**Project 3 Report**

There were a number of problems I encountered with project 3. The first function, isPathWellFormed, was fairly simply and I did not have many real problems with it, but the other two were quite tricky. One of the hardest parts of coding followSegment was just determining what structure to use; it ended up as a series of if statements and a for loop, not the most efficient code in the world. I feel like it could have been optimized somewhat, mainly because as I remembered different tests I simply added them onto the top of the existing if statements. By the end there was around 7 or so, all just to check for errors. The hardest part of coding followPath was testing – I found it unable to test until I had finished writing the entire function, and by then the numerous bugs had piled up considerably. I really should have done the testing incrementally as doing it all at once simply made things take longer. Simply getting the logic right (checking the next digits if a letter is parsed, skipping if a digit is parsed) took a lot of mistakes before I found a solution that worked adequately.

My program is designed thusly:

* isPathWellFormed is mainly just a series of if statements checking for errors. The program first parses the entire input into lowercase, and then checks that there are no spaces, that all characters are alphanumeric, that there are never 2 letters in a row, and that there are never 3 digits in a row (through the use of a “counter” variable.) If the input passes all the tests, the function returns 1; if it fails it returns 0.
* followSegment first runs another series of error checking: rows and columns must be in the correct domain, maxSteps must be positive, and the starting position cannot be inside a wall. If any of these tests fail, the program returns a -1; otherwise it continues on and adds up the total steps. It does so by first checking *dir* for which direction the steps are headed, then executing a *for* loop which counts up to maxSteps and moves the position in the direction indicated by one step until it hits a wall or the edge of the maze. When it either reaches maxSteps, hits a wall, or reaches the edge of the maze, the loop stops and the program returns the amount of steps taken.
* followPath declares a couple variables to keep track of the current position of the robot, and checks that isPathWellFormed evaluates to true before continuing on (if not, it returns a 3.) First, the program checks if a character is a letter; if this is true, it then checks if the letter is followed by 1 digit or 2 (0 and 3+ are not possible because to reach this point the Boolean must have evaluated to true.) If it is followed by one digit, it sets the number of maxSteps to be plugged into followSegment is equal to this digit, converted to an int by a small function. If it is followed by 2 digits, it sets maxSteps to be the first digit times 10, plus the second digit. Then it plugs maxSteps and dir (the parsed letter) into followSegment, along with the current position, and moves the robot the number of steps returned by the function in the direction indicated. It also checks to see if the number of steps plugged in is the same coming out; if it is not, the robot must have hit a wall and it sets the wall check Boolean to true. At this point the loop breaks and goes on to the next parsed character, skipping digits and evaluating letters until reaching the end of the string. When the loop finishes, the program checks the final position – if the robot is at the intended end position, it returns 0; if not, it returns 1; if it hit a wall at some point, it returns 2.

To test this program, I wrote a main function that created a 12x12 maze. I varied the start positions and end positions, created a bunch of walls and moved them around, and tested various inputs to see what the program returned. My debug code would ask for a path, assign the path first to be checked by the Boolean, then to followPath. My output consisted of the syntax test result, the incremental position after each step in the loop of followPath, a step checker in followSegment, the final position, and the final output of followPath (0, 1, 2, or 3.) This made testing simple: just plug in an input and one could get the result of all 3 functions at once.